This is my prototype for MakeC:

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

double \*make\_vector(int N);

double \*MakeC(int N)

{

double \*C,pi=3.14159265358979323846;

int i,j,k;

if (N%3==0)

{

if (N>3)

C=make\_vector(N/2-1);

else

C=make\_vector(N);

C[0]=cos(pi/3);

C[1]=cos(pi/6);

k=2;

for (i=12; i<=N; i\*=2)

{

for (j=1; j<=(i/2-1); j+=2)

{

C[k]=cos(j\*pi/i);

k++;

}

}

}

else

{

k=0;

C=make\_vector(N/2-1);

for (i=4; i<=N; i\*=2)

{

for (j=1; j<=(i/2-1); j+=2)

{

C[k]=cos(j\*pi/i);

k++;

}

}

}

return(C);

}

Here are my prototypes to make CN, DN and EN respectively (they are used only to test whether my fast DCT is correct):

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

double \*\*make\_matrix(int NR, int NC);

double \*\*MakeCN(int N)

{

double \*\*CN,pi=3.14159265358979323846;

int i,j;

CN=make\_matrix(N,N);

for (i=0; i<=N; i++)

{

for (j=0; j<=N; j++)

CN[i][j]=cos(i\*j\*pi/N);

}

return(CN);

}

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

double \*\*make\_matrix(int NR, int NC);

double \*\*MakeDN(int N)

{

double \*\*DN,pi=3.14159265358979323846;

int i,j;

DN=make\_matrix(N,N);

for (i=1; i<=N; i++)

{

for (j=0; j<=N-1; j++)

DN[i][j]=cos((2\*i-1)\*j\*pi/(2\*N));

}

return(DN);

}

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

double \*\*make\_matrix(int NR, int NC);

double \*\*MakeEN(int N)

{

double \*\*EN,pi=3.14159265358979323846;

int i,j;

EN=make\_matrix(N,N);

for (i=1; i<=N; i++)

{

for (j=1; j<=N; j++)

EN[i][j]=cos((2\*i-1)\*(2\*j-1)\*pi/(4\*N));

}

return(EN);

}

And here are my make\_vector and make\_matrix prototypes, which differ slightly from my previous project:

#include<malloc.h>

#include<stdlib.h>

double \*make\_vector(int N)

{

double \*M; int n;

M=(double \*)malloc((N+1)\*sizeof(double));

for (n=1; n<=N; n++) M[n]=M[n-1]+N+1;

return(M);

}

#include<malloc.h>

#include<stdlib.h>

double \*\*make\_matrix(int NR, int NC)

{

double \*\*M; int n;

M=(double \*\*)malloc((NR+1)\*sizeof(double \*));

M[0]=(double \*)malloc(((NR+1)\*(NC+1))\*sizeof(double));

for (n=1; n<=NR; n++) M[n]=M[n-1]+NC+1;

return(M);

}

My FastCosine, which only calls FastCN:

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

void \*FastCN(double \*x, double \*y, double \*w, int N, int skip, double \*C);

void FastCosine(double \*x, double \*y, double \*w,

double \*C, int N, int skip)

{

y[0]=y[0]/2;

y[N\*skip]=y[N\*skip]/2;

FastCN(x,y,w,N,skip,C);

}

My FastCN, which splits  into  and  parts, then calls FastCN and FastDN. When N is small enough, we simulate the matrix-vector calculation without actually creating the matrix CN.

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

double \*\*MakeCN(int N);

void \*FastDN(double \*x, double \*y, double \*w, int N, int skip, double \*C);

void \*FastCN(double \*x, double \*y, double \*w, int N, int skip, double \*C)

{

int i;

if (N>3)

{

w[0]=y[0]+y[N];

w[N/2]=y[N/2];

w[N/2+1]=y[0]-y[N];

for (i=1;i<N/2;i++)

{

w[i]=y[i]+y[N-i];

w[i+N/2+1]=y[i]-y[N-i];

}

for (i=0;i<=N;i++)

y[i]=w[i];

N=N/2;

skip=skip\*2;

FastCN(x,y,w,N,skip,C);

FastDN(x+N,y+N,w,N,skip,C);

for (i=0;i<=N\*2;i++)

w[(i\*2)%(N\*2+1)]=x[i];

for (i=0;i<=N\*2;i++)

x[i]=w[i];

}

else if (N==2)

{

x[0]=y[0]+y[1]+y[2];

x[1]=y[0]-y[2];

x[2]=y[0]-y[1]+y[2];

}

else

{

x[0]=y[0]+y[1]+y[2]+y[3];

x[1]=y[0]+y[1]/2-y[2]/2-y[3];

x[2]=y[0]-y[1]/2-y[2]/2+y[3];

x[3]=y[0]-y[1]+y[2]-y[3];

}

return(0);

}

My FastDN, which splits  into  and  parts, then calls FastDN and FastEN. When N is small enough, we simulate the matrix-vector calculation without actually creating the matrix DN.

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

double \*\*MakeDN(int N);

void \*FastEN(double \*x, double \*y, double \*w, int N, int skip, double \*C);

void \*FastDN(double \*x, double \*y, double \*w, int N, int skip, double \*C)

{

int i;

if (N>3)

{

for (i=1;i<=N/2;i++)

{

w[i]=y[2\*i-1];

w[N/2+i]=y[2\*i];

}

for (i=1;i<=N;i++)

y[i]=w[i];

N=N/2;

skip\*=2;

FastDN(x,y,w,N,skip,C);

FastEN(x+N,y+N,w,N,skip,C);

for (i=1;i<=N;i++)

{

w[i]=x[i]+x[i+N];

w[N\*2-i+1]=x[i]-x[i+N];

}

for (i=1;i<=N\*2;i++)

x[i]=w[i];

}

else if (N==2)

{

x[1]=y[1]+y[2]/sqrt(2);

x[2]=y[1]-y[2]/sqrt(2);

}

else

{

x[1]=y[1]+sqrt(3)\*y[2]/2+y[3]/2;

x[2]=y[1]-y[3];

x[3]=y[1]-sqrt(3)\*y[2]/2+y[3]/2;

}

return(0);

}

My FastEN, which splits  into  and  parts, then calls FastDN twice. When N is small enough, we simulate the matrix-vector calculation without actually creating the matrix EN.

#include<malloc.h>

#include<stdlib.h>

#include<math.h>

double \*\*MakeEN(int N);

void \*FastDN(double \*x, double \*y, double \*w, int N, int skip, double \*C);

void \*FastEN(double \*x, double \*y, double \*w, int N, int skip, double \*C)

{

double pi=3.14159265358979323846,n;

int i,k,l;

if (N>3)

{

w[1]=y[1];

w[N/2+1]=y[N];

for (i=2; i<=N/2; i++)

{

w[i]=y[(i-1)\*2]+y[2\*i-1];

w[N-i+2]=y[(i-1)\*2]-y[2\*i-1];

}

for (i=1;i<=N;i++)

y[i]=w[i];

N=N/2;

skip\*=2;

FastDN(x,y,w,N,skip,C);

FastDN(x+N,y+N,w,N,skip,C);

k=N\*2-2;

l=N\*4-1;

for (i=1;i<=N;i+=2)

{

w[i]=x[i]\*C[k+i]+x[N+i]\*C[l-i];

w[N\*2+1-i]=x[i]\*C[l-i]-x[N+i]\*C[k+i];

}

for (i=2;i<=N;i+=2)

{

w[i]=x[i]\*C[k+i]-x[N+i]\*C[l-i];

w[N\*2+1-i]=x[i]\*C[l-i]+x[N+i]\*C[k+i];

}

for (i=1;i<=N\*2;i++)

x[i]=w[i];

}

else if (N==2)

{

n=cos(3\*pi/8);

x[1]=y[1]\*C[1]+y[2]\*n;

x[2]=y[1]\*n-y[2]\*C[1];

}

else

{

x[1]=y[1]\*C[2]+y[2]\*C[3]+y[3]\*C[4];

x[2]=y[1]\*C[3]-y[2]\*C[3]-y[3]\*C[3];

x[3]=y[1]\*C[4]-y[2]\*C[3]+y[3]\*C[2];

}

return(0);

}

Here are some additional prototypes which I’ll be needing to test out my main.

#include<malloc.h>

#include <stdlib.h>

void free\_matrix(double \*\*M)

{

free(M[0]);

free(M);

}

#include<malloc.h>

#include <stdlib.h>

void free\_vector(double \*M)

{

free(M);

}

#include<malloc.h>

#include<stdlib.h>

double \*make\_vector(int N);

double \*make\_random\_vector(int N)

{

double \*M; int i;

M=make\_vector(N);

for (i=0; i<=N; i++)

M[i]=(double)rand()/(double)RAND\_MAX;

return(M);

}

#include<malloc.h>

#include <stdlib.h>

double \*make\_vector(int N);

double \*mat\_vec\_mult(double \*\*M, double \*N, int n)

{

double \*A;

int i,j;

A=make\_vector(n);

for (i=0;i<=n;i++)

{

A[i]=0;

for (j=0;j<=n;j++)

A[i]=A[i]+N[j]\*M[i][j];

}

return(A);

}

#include<malloc.h>

#include <stdlib.h>

#include<omp.h>

double \*\*make\_matrix(int NR, int NC);

double \*\*matrix\_mult

(double \*\*B, double \*\*C, int L, int M)

{

double \*\*A;

int i,j,k;

A=make\_matrix(L,M);

for (i=1;i<=L;i++)

{

for (j=1;j<=M;j++)

A[i][j]=0;

}

for (i=1;i<=L;i++)

{

for (j=1;j<=M;j++)

{

for (k=1;k<=L;k++)

A[i][j]=A[i][j]+B[i][k]\*C[k][j];

}

}

return(A);

}

#include<math.h>

#include<malloc.h>

#include <stdlib.h>

double \*make\_vector(int N);

double norm(double \*N, int n)

{

double A;

int i;

A=0;

for (i=0;i<=n;i++)

{

A=A+N[i]\*N[i];

}

A=sqrt(A);

return(A);

}

#include<stdio.h>

#include<malloc.h>

#include <stdlib.h>

void print\_vector(double \*M, int N)

{

int i;

for (i=0; i<=N; i++)

printf("%12g\n",M[i]);

printf("\n\n");

}

#include<malloc.h>

#include <stdlib.h>

double \*make\_vector(int N);

double \*vec\_sub(double \*B, double \*C, int n)

{

double \*A;

int i;

A=make\_vector(n);

for (i=0;i<=n;i++)

{

A[i]=B[i]-C[i];

}

return(A);

}

Here is my main that tests the DCT from 2 to 16, by multiples of 2

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<time.h>

double \*MakeC(int N);

void print\_vector(double \*M, int N);

void print\_matrix(double \*\*M, int NR, int NC);

double \*\*MakeCN(int N);

double \*make\_vector(int N);

void FastCosine(double \*x, double \*y, double \*w,

double \*C, int N, int skip);

double \*make\_random\_vector(int N);

double \*mat\_vec\_mult(double \*\*M, double \*N, int n);

void free\_vector(double \*M);

void free\_matrix(double \*\*M);

double \*vec\_sub(double \*B, double \*C, int n);

double norm(double \*N, int n);

int main(void)

{

double \*C, \*x, \*y, \*w, \*A, \*\*CN, then, now, diff;

int N, skip, i, j;

printf("Name:\t\tJialu Tu\n");

printf("Email Address:\tjialu.tu06@imperial.ac.uk\n");

printf("Course Code:\tG100\n");

printf("Module Code:\tM3SC\n");

printf("Date:\t\t%s\n",\_\_DATE\_\_);

printf("Time:\t\t%s\n\n",\_\_TIME\_\_);

for (N=2; N<=16; N\*=2)

{

skip=1;

y=make\_random\_vector(N);

x=make\_vector(N);

w=make\_vector(N);

C=MakeC(N);

for (i=0;i<=N;i++)

w[i]=y[i];

w[0]=w[0]/2;

w[N]=w[N]/2;

CN=MakeCN(N);

A=make\_vector(N);

A=mat\_vec\_mult(CN,w,N);

FastCosine(x,y,w,C,N,skip);

printf("N=%d\n",N);

print\_vector(x,N);

print\_vector(A,N);

printf("\n");

free\_vector(C);free\_vector(x);

free\_vector(y);free\_vector(w);

free\_matrix(CN);

}

return(0);

}

And this is the result, the top is the result from my fast DCT and the bottom is by just doing matrix vector calculation. As we can see here, the results are the same.

Name: Jialu Tu

Email Address: jialu.tu06@imperial.ac.uk

Course Code: G100

Module Code: M3SC

Date: Apr 26 2009

Time: 20:42:54

N=2

0.660863

-0.0960265

-0.466308

0.660863

-0.0960265

-0.466308

N=4

2.26753

0.12236

0.372478

-0.209582

0.396924

2.26753

0.12236

0.372478

-0.209582

0.396924

N=8

3.77979

1.08199

-0.685344

0.418425

0.68952

0.496699

0.178584

-0.534236

-0.488342

3.77979

1.08199

-0.685344

0.418425

0.68952

0.496699

0.178584

-0.534236

-0.488342

N=16

6.20811

-0.768196

0.378568

0.804565

-0.837779

-1.71671

-0.577422

0.163412

0.384228

0.972758

0.180313

1.29111

0.204398

0.102406

-0.0598306

-0.800032

0.17954

6.20811

-0.768196

0.378568

0.804565

-0.837779

-1.71671

-0.577422

0.163412

0.384228

0.972758

0.180313

1.29111

0.204398

0.102406

-0.0598306

-0.800032

0.17954

Press any key to continue . . .

To test out my result to deal with cases N multiples of 3, I have used a different approach for my main.c, which is calculation the norm of the difference of the vector calculation from my transform and the direct matrix multiplication approach.

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#include<time.h>

double \*MakeC(int N);

void print\_vector(double \*M, int N);

void print\_matrix(double \*\*M, int NR, int NC);

double \*\*MakeCN(int N);

double \*make\_vector(int N);

void FastCosine(double \*x, double \*y, double \*w,

double \*C, int N, int skip);

double \*make\_random\_vector(int N);

double \*mat\_vec\_mult(double \*\*M, double \*N, int n);

void free\_vector(double \*M);

void free\_matrix(double \*\*M);

double \*vec\_sub(double \*B, double \*C, int n);

double norm(double \*N, int n);

int main(void)

{

double \*C, \*x, \*y, \*w, \*A, \*\*CN, then, now, diff;

int N, skip, i, j;

printf("Name:\t\tJialu Tu\n");

printf("Email Address:\tjialu.tu06@imperial.ac.uk\n");

printf("Course Code:\tG100\n");

printf("Module Code:\tM3SC\n");

printf("Date:\t\t%s\n",\_\_DATE\_\_);

printf("Time:\t\t%s\n\n",\_\_TIME\_\_);

for (N=3; N<=5000; N\*=2)

{

skip=1;

y=make\_random\_vector(N);

x=make\_vector(N);

w=make\_vector(N);

C=MakeC(N);

for (i=0;i<=N;i++)

w[i]=y[i];

w[0]=w[0]/2;

w[N]=w[N]/2;

CN=MakeCN(N);

A=make\_vector(N);

A=mat\_vec\_mult(CN,w,N);

FastCosine(x,y,w,C,N,skip);

printf("N=%d\n",N);

printf("%g\n",norm(vec\_sub(A,x,N),N));

printf("\n");

free\_vector(C);free\_vector(x);

free\_vector(y);free\_vector(w);

free\_matrix(CN);

}

return(0);

}

And our result, as we can see here, the norm of the difference is very small.

Name: Jialu Tu

Email Address: jialu.tu06@imperial.ac.uk

Course Code: G100

Module Code: M3SC

Date: Apr 26 2009

Time: 20:58:33

N=3

1.00074e-016

N=6

1.50316e-015

N=12

2.6273e-015

N=24

1.5235e-014

N=48

4.66494e-014

N=96

2.6065e-013

N=192

1.20003e-012

N=384

4.04982e-012

N=768

1.8407e-011

N=1536

6.1469e-011

N=3072

2.6107e-010

Press any key to continue . . .